Apple Assembly Line

$1.50

Volume 3 -- Issue 11 August, 1983

In This Issue...

Mailing AAL

Let's review how AAL is mailed, when you should expect to receive it, and what to do about it when you don't. Most of you get your newletter by Bulk Mail, which is a little erratic. You should receive your issue around the third week of each month, but don't start worrying until the end of the month. If you haven't received an issue by the end of the month, call or write and we'll send a replacement. The Post Office does not forward Bulk Mail, so make certain to tell us if you move. Those of you who have First Class Mail subscriptions should receive your issue around the tenth of the month, and certainly before the twentieth.

The number in the upper right corner of your mailing label is the expiration date of your subscription. If that number is 8308, you're holding your last issue and better renew now. We send out postcards when your subscription is about to expire, and when it has expired. All you have to do is send us a check, or phone with a charge card number, and we'll keep your AAL coming.

Using Auxiliary Memory in the //e.....David C. Johnson
Ridgefield, CT

When I bought my Apple //e (3 days after they became available!), I also got the Extended 80-Column Text Card. I wanted it both to have 80 column text capability and a full complement of Apple Computer Inc. supported memory. However, Apple only supplied two small subroutines in ROM and incomplete (but otherwise excellent) documentation in their manuals, in "support" of the auxiliary memory.

I say "incomplete" because two I/O locations that I used in my program are not mentioned (in English anyway) anywhere in the manuals except in the listings of the 80-column firmware. The two I/O locations are $C011 & $C012 which I call READ.BSR.BANK & READ.BSR.RAM.READ. Apple evidently intends to let software developers determine how the auxiliary memory is to be used.

Well here goes: my program is called "USE.AUXMEM". This program allows you to access the "other" 64K in a manner most Apple users should already be familiar with: monitor commands.

The simplest way to see what I mean is to type in & assemble the program (not so simple), type :"MGO G", :"PR#3" and then :"$^Y" (that is control-Y). You will get a bell and the monitor's prompt. Any monitor commands you type now will "use" the auxiliary memory. Try these now:

```
*3D0:55
*3D0      (double nickels, right?)
*^Y       (back to SCASM!)
:$3D0     (a $4C!)
```

You should note that control-Y while using the auxiliary memory returns to main memory with everything as it was. Now try these:

```
:$^Y 3D0
*3D0 ^Y
```

After the second control-Y returned to main memory, SCASM finished the first command line!

The reason I had you type :"PR#3" before is quite simple: things don't all work right without the 80 column firmware active; specifically, right-arrow & escape functions. You can also type "escape 4" if you don't want 80 columns.

But wait a minute, if you read the 80-column firmware listing (carefully), you know that it does NOT work with the auxiliary memory enabled (as doesn't the regular 40-column firmware), so how is this all working? Well, the I/O hooks in the auxiliary memory zero page point to routines in USE.AUXMEM which switch to main memory, perform the I/O, switch back to auxiliary memory, and return to the monitor. The monitor executes its commands between I/O calls while auxiliary memory is enabled. These switchings also change the bank switched memory state.

S-C Macro Assembler (the best there is!).....$80.00
S-C Macro Assembler Version 1.1 Update.....$12.50

S-C Cross Reference Utility.....$20.00
S-C Cross Reference Utility with Complete Source Code.....$50.00

S-C Word Processor.....$50.00
  As is, with fully commented source code. Needs S-C Macro Assembler.
Applesoft Source Code on Disk.....$50.00
    Very heavily commented. Requires Applesoft and S-C Assembler.
ES-CAPE: Extended S-C Applesoft Program Editor.....$60.00

AAL Quarterly Disks.....each $15.00
    Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980 QD#2: Jan-Mar 1981 QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981 QD#5: Oct-Dec 1981 QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982 QD#8: Jul-Sep 1982 QD#9: Oct-Dec 1982
    QD#10: Jan-Mar 1983 QD#11: Apr-Jun 1983

Double Precision Floating Point for Applesoft.....$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)..... $79.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).....$49.00

The Visible Computer: 6502 (Software Masters).....(reg. $50.00) $45.00
Super Disk Copy III (Sensible Software).....(reg. $30.00) $27.00
Amper-Magic (Anthro-Digital).....(reg. $75.00) $67.50
Amper-Magic Volume 2 (Anthro-Digital).....(reg. $35.00) $30.00
Quick-Trace (Anthro-Digital).....(reg. $50.00) $45.00
DISASM Dis-Assembler (RAK-Ware).....$30.00

Blank Diskettes (with hub rings).....package of 20 for $45.00
Small 3-ring binder with 10 vinyl disk pages and disks.....$36.00
Vinyl disk pages, 6"x8.5", hold one disk each.....10 for $6.00
Diskette Mailing Protectors.....10-99: 40 cents each
                                100 or more: 25 cents each
ZIF Game Socket Extender.....$20.00
Shift-Key Modifier.....$15.00
Lower-Case Display Encoder ROM.....$25.00
    Only Revision level 7 or later Apples.
WICO Track Ball.....($89.95) $80.00
STB-80 80-column Display Board (STB Systems).....($249.00) $225.00
STB-128 128K RAM Card (STB Systems).....($399.00) $350.00

Grappler+ Printer Interface (Orange Micro).....($175.00) $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00) $150.00
Buffered Grappler+ NEW!! Interface and 16K Buffer.....($239.00) $200.00

Books, Books, Books.....compare our discount prices!
    "THE Book of Apple Software 1983 (with supplement)...($24.90) $18.00
    "The Apple][Circuit Description", Gayler.....($22.95) $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.....($17.95) $17.00
    "Incredible Secret Money Machine", Lancaster.....($7.95) $7.50
    "Micro Cookbook, vol. 1", Lancaster.....($15.95) $15.00
    "Beneath Apple DOS", Worth & Lechner.....($19.95) $18.00
    "Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36.00
    "Apple Graphics & Arcade Game Design", Stanton.....($19.95) $18.00
    "Assembly Lines: The Book", Roger Wagner.....($19.95) $18.00
    "What's Where in the Apple", Second Edition.....($24.95) $23.00
    "What's Where Guide" (updates first edition).....($9.95) $9.00
    "6502 Assembly Language Programming", Leventhal.....($18.95) $18.00
    "6502 Subroutines", Leventhal.....($17.95) $17.00
  Add $1.50 per book for US postage. Foreign orders add postage needed.

Whatever Else You Need.....Call for Our Low Prices

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We take Master Charge, VISA and American Express ***

The USE.AUXMEM program has two other control-Y commands. They implement the crossbank subroutines AUXMOVE & XFER (supplied in ROM) as monitor commands. See the comments at the top of the source listing for their syntax.

About Some //e Monitor Bugs...

One routine, USE.AUXMEM.CONTROL.Y.HANDLER, deserves a special note. It compensates for a bug in the Apple //e version of the monitor: when parsing a control-Y command the ASCII string "Bryan" at $FEC5 is executed as instructions prior to JMPing to USRADR ($03F8). This bug has a long history.

In the original Apple monitor the CHRSRCH loop ($FF78 - $FF81) scans the CHRTBL ($FFCC - $FFE2) from end to beginning, which matches the $B2 at $FFCD causing TOSUB ($FFBE - $FFCB) to load the $C9 at $FFE4 and RTS to USR ($FECA) which is a JMP USRADR ($03F8).

Things started to go astray when the autostart ROM was created, and the Apple II Plus. To make room for new features, (like printing "APPLE][" at the top of the screen on power up, and like the escape-IJKM cursor motion), the TRACE and STEP commands were removed. To disable the entries for Trace and Step in CHRTBL, the bytes for "T" and "S" were changed: ($FFCF:B2 & $FFD2:B2, also $FFE9:C4). Locations $FEC5 - $FEC9, immediately prior to USR, were changed to NOPs.

Unfortunately, someone forgot that CHRTBL is searched from end to beginning, causing a control-Y command to be matched with the $B2 at $FFD2, corresponding to the branch address in SUBTBL at $FFE9. So when you type a control-Y command the monitor branches to $FFC5 and executes the 5 NOPs. If $FFE9 had been changed to $C9 instead of $C4, everything would have still been fine.

Executing 5 NOPs is not a bad bug. But when the Apple //e monitor was created those 5 NOPs were replaced by the string "Bryan". In hex it is C2 F2 F9 E1 EE. The 6502 instruction set does not include a definition for $C2, but after a little investigation, or after reading Bob Sander-Cederlof's article in AAL March 1981, you find out that $C2 acts like a two-byte NOP. The "r" is skipped over. The "yan", however, is a SBC $EEE1,Y instruction.

The USE.AUXMEM.CONTROL.Y.HANDLER uses the passed contents of the A & X registers to decide which of the three control-Y commands you've typed. The SBC $EEE1,Y changes the A register so its contents must be reconstructed. The reconstruction is further complicated by the fact that the monitor leaves the carry flag set when it RTS's to $FEC5, while the S-C Assembler and Mini-Assembler leave the carry flag clear.

To restore the A register to its proper value you must set the carry to the complement of the value that it was set to prior to the SBC $EEE1,Y then execute ADC $EEE1,Y.

The Apple //e 80 column firmware also contains a bug. Because of the $11 at $C92A, the key sequence "ESCape ^L" causes a RTS to $4CCE. Location $C92A should contain a $10. This bug can be used to advantage if you feel like adding a secret command to your own software. Just be certain you have the code for your command starting at $4CCE, and that you are running in 80-column mode. Then whenever you type control-L in the escape mode (cursor is an inverse plus) your code will be executed.

I hope all of you enjoy using your auxiliary memory as much as I do.

Last Minute Note: David just called to report yet another oddity in the //e ROMs. In 40-column ESCape mode the (, 5, *, and + keys duplicate the arrow keys. That is, "ESC 5" moves the cursor right one space, just like ESC right arrow. This is a little bit weird, but it doesn't seem to hurt anything. The effect is caused by an unnecessary AND #$DF instruction at $C26E.

```
                1000 *SAVE JOHNSON'S USE AUXMEM
                1010 *-------------------------------
                1020 * SWITCH.MIND Command:  ^Y
                1030 *
                1040 *   When in main bank, enters monitor in
                1050 *   auxmem BSR (hooks I/O through main
                1060 *   and brings USE.AUXMEM to auxmem too)
                1070 *   When in aux bank, returns to main bank
                1080 *   Best used w/80 column firmware active
                1090 *-------------------------------
                1100 * USE.AUXMOVE Command:  DEST<SOURCE.ENDˆY{CARRY}
                1110 *
                1120 *   DEST       = Destination in one bank
                1130 *   SOURCE     = Start in other bank
                1140 *   END        = End in other bank
                1150 *   CARRY      = Direction of move
                1160 *                (1 = Main Ram-->Card Ram)
                1170 *                (0 = Card Ram-->Main Ram)
                1180 *   DEST, SOURCE, & END must be: >=$0200 & <=$BFFF
                1190 *-------------------------------
                1200 * USE.XFER Command:  ADDRESSˆY{CARRY}{OVERFLOW}
                1210 *
                1220 *   ADDRESS    = Transfer address
                1230 *   CARRY      = Desired 48K Bank ($0200 - $BFFF)
                1240 *                (1 = Use 48K in Card Ram)
                1250 *                (0 = Use 48K in Main Ram)
                1260 *   OVERFLOW   = Desired ZP/STK/BSR
                1270 *                (1 = Use ZP/STK/BSR in Card Ram)
                1280 *                (0 = Use ZP/STK/BSR in Main Ram)
                1290 *   If using USE.XFER from auxmem, routine in main mem
                1300 *   MUST LDX BANK.SP.SAVE, TXS if it uses the stack at all
                1310 *-------------------------------
0028-           1320 MON.BASL           .EQ $28,$29
0034-           1330 MON.YSAV           .EQ $34
0036-           1340 MON.CSWL           .EQ $36,$37
0038-           1350 MON.KSWL           .EQ $38,$39
003C-           1360 MON.A1             .EQ $3C,$3D   Source,Address
003E-           1370 MON.A2             .EQ $3E,$3F   End
0042-           1380 MON.A4             .EQ $42,$43   Dest
0048-           1390 MON.STATUS         .EQ $48
                1400 *-------------------------------
0200-           1410 IN                         .EQ $0200 - $02FF
03CC-           1420 BANK.X.SAVE                .EQ $03CC
03CD-           1430 BANK.BSR.BANK.SAVE         .EQ $03CD
03CE-           1440 BANK.BSR.RAM.READ.SAVE     .EQ $03CE
03CF-           1450 BANK.SP.SAVE               .EQ $03CF
03ED-           1460 TRANSFER                   .EQ $03ED,$03EE
03F0-           1470 MON.BRKV                   .EQ $03F0,$03F1
03F8-           1480 USRADR                     .EQ $03F8 - $03FA
03FB-           1490 NMI                        .EQ $03FB - $03FD
03FE-           1500 MON.IRQLOC                 .EQ $03FE,$03FF
```

```
                    1510 *--------------------------------
C002-               1520 READ.MAIN.RAM              .EQ $C002
C003-               1530 READ.AUX.RAM               .EQ $C003
C004-               1540 WRITE.MAIN.RAM             .EQ $C004
C005-               1550 WRITE.AUX.RAM              .EQ $C005
C008-               1560 USE.MAIN.ZP.STK.BSR        .EQ $C008
C009-               1570 USE.AUX.ZP.STK.BSR         .EQ $C009
C011-               1580 READ.BSR.BANK              .EQ $C011
C012-               1590 READ.BSR.RAM.READ          .EQ $C012
C013-               1600 READ.RAM.READ.STATUS       .EQ $C013
C014-               1610 READ.RAM.WRITE.STATUS      .EQ $C014
C016-               1620 READ.ZP.STK.BSR.STATUS     .EQ $C016
C080-               1630 BSR.2.RAM.READ.ONLY        .EQ $C080
C081-               1640 BSR.2.ROM.READ.RAM.WRITE   .EQ $C081
C082-               1650 BSR.2.ROM.READ.ONLY        .EQ $C082
C083-               1660 BSR.2.RAM.READ.RAM.WRITE   .EQ $C083
C088-               1670 BSR.1.RAM.READ.ONLY        .EQ $C088
C089-               1680 BSR.1.ROM.READ.RAM.WRITE   .EQ $C089
C08A-               1690 BSR.1.ROM.READ.ONLY        .EQ $C08A
C08B-               1700 BSR.1.RAM.READ.RAM.WRITE   .EQ $C08B
                    1710 *--------------------------------
C311-               1720 AUXMOVE                    .EQ $C311
C314-               1730 XFER                       .EQ $C314
F800-               1740 MONITOR                    .EQ $F800 - $FFFF
FA59-               1750 MON.OLDBRK                 .EQ $FA59
FBDD-               1760 BEEP                       .EQ $FBDD
FD0C-               1770 MON.RDKEY                  .EQ $FD0C
FD8B-               1780 MON.JSR.CLREOL             .EQ $FD8B - $FD8D
FDED-               1790 MON.COUT                   .EQ $FDED
FF65-               1800 MON                        .EQ $FF65
                    1810 *--------------------------------
                    1820                 .OR $0803
                    1830 USE.AUXMEM
                    1840 G
0803- 4C 12 08      1850                 JMP CONNECT.CONTROL.Y
                    1860 JMP.TO.RETURN.TO.MAIN
0806- 4C 25 09      1870                 JMP RETURN.TO.MAIN
                    1880 JMP.TO.RETURN.TO.AUX
0809- 4C 19 09      1890                 JMP RETURN.TO.AUX
                    1900 JMP.TO.SAVE.BSR.STATE
080C- 4C 61 09      1910                 JMP SAVE.BSR.STATE
                    1920 JMP.TO.RESTORE.BSR.STATE
080F- 4C 32 09      1930                 JMP RESTORE.BSR.STATE
                    1940 *--------------------------------
                    1950 CONNECT.CONTROL.Y
0812- A9 08         1960                 LDA /USE.AUXMEM.CONTROL.Y.HANDLER
0814- 8D FA 03      1970                 STA USRADR+2
0817- A9 22         1980                 LDA #USE.AUXMEM.CONTROL.Y.HANDLER
0819- 8D F9 03      1990                 STA USRADR+1
081C- A9 4C         2000                 LDA #$4C        JMP
081E- 8D F8 03      2010                 STA USRADR
0821- 60            2020                 RTS
                    2030 *--------------------------------
                    2040 USE.AUXMEM.CONTROL.Y.HANDLER
                    2050 * Reconstruct monitor mode byte
                    2060 * after "Bryan" messed with it
                    2070 * ("Br" is NOPish)
0822- 48            2080                 PHA
0823- AD 00 02      2090                 LDA IN
0826- C9 A4         2100                 CMP #"$"
                    2110 * Branch w/Carry set causa S-C or Mini-Asm
0828- F0 01         2120                 BEQ .1
082A- 18            2130                 CLC
082B- 68            2140 .1              PLA
                    2150 * These lines are for you Bryan
082C- 79            2160                 .DA #'y'
082D- E1 EE         2170                 .AS -'an'       Builds SBC $EEE1,Y
                    2180 * Check for user specified address
082F- E0 01         2190                 CPX #$01
0831- D0 57         2200                 BNE SWITCH.MIND
0833- A8            2210                 TAY
                    2220 * Lesser complex is USE.XFER
0834- F0 12         2230                 BEQ USE.XFER
                    2240 * Most complex is USE.AUXMOVE
```

GOT A FUNNY DISK?

... WANT TO KNOW MORE ABOUT IT?

Then you need the

(Confidential Information Advisors)

CAN YOU ...

- ★ **edit normal or protected disks?**
- ★ **quickly find and recover any intact file, however badly the disk is corrupted?**
- ★ **list programs directly from any disk - protected or not?**
- ★ **examine textfiles directly from any disk - protected or not?**
- ★ **analyse the formatting of normal or protected disks?**
- ★ **decrypt commercial software - or encrypt your own?**
- ★ **rapidly auto-search normal or protected disks for anything you like?**
- ★ **understand & use the latest copy protection methods?**
- ★ **use your Apple as a powerful document retrieval system?**
- ★ **make use of an exhaustive knowledge of disk lore?**

YOU CAN NOW — with a little help from these 5 sophisticated disk utilities:

TRICKY DICK examines, records, deletes, and edits. It can: (1) read individual sectors from normal and most protected disks, (2) list their contents in BASIC, assembler, ASCII, or hex, (3) edit them, (4) write them back to the disk. Tricky Dick cunningly bypasses most protection systems, allowing you to work on disks with nonstandard formatting, half-tracks, and altered DOS marks. It is also a chief executive program that directs the following undercover agents:

THE LINGUIST reads in a trackful of raw data for your scrutiny, translates all the address information, and allows you to inspect the track's formatting. It also translates all 3 types of DOS encoding (6 & 2, 5 & 3, 4 & 4), and works with Tricky Dick to list and examine programs or textfiles on any protected disk. You can use The Linguist to recover valuable files from blown disks, improve your programming skills by studying commercial software, and analyse standard or altered formatting.

THE TRACER rapidly searches normal and most protected disks for up to six strings of your choice simultaneously (specified in ASCII or hex). The Tracer also verifies disk formatting, and sniffs out all hidden catalog or VTOC sectors. When it finds something, it transfers control to Tricky Dick and puts the cursor over the object of your search. A few further keystrokes allow you to make any necessary changes and write the sector back to the disk.

THE CODE BREAKER keeps your programs and textfiles from prying eyes by enabling you to translate them into a "secret code" during disk storage. This utility also deciphers encrypted commercial programs, allowing you to use Tricky Dick to read, list, and edit software never before accessible to any disk utility.

THE TRACKER closely shadows the disk drive arm, carefully recording all its movements and operations. The Tracker's job is to display, on either your screen or printer, a list of every track and sector accessed during a LOAD, RUN, SAVE, or any other DOS operation. This utility also tells you exactly where a read or write occurred during any disk access. Use The Tracker's services to locate the precise trouble spots on a clobbered disk, to determine sector skew patterns, to discover the location of hidden "nibble-count" tracks on protected disks, and to learn much more about how DOS works. You'll be surprised to see just exactly where the disk arm really does go!

What's more, you get permanent access to:

THE CIA FILES, a 50,000+ word book designed to turn you into a disk expert. In addition to complete instructions for the 5 CIA utilities, the book contains an easy-to-follow hand-holding tutorial (written in plain English!) on all aspects of the Apple disk. Using the CIA utilities as your personal guides, you progress step-by-step to total disk mastery. You'll acquire a wealth of skills and information relating to disk repair and file recovery, DOS patches, copy protection, disk formatting, program encryption, and other vital topics. Much of the material has never before appeared in print.

All programs are UNPROTECTED, and hence can be copied, listed, and modified at will. (special patches are described in the manual). They require one drive, DOS 3.3, and 48K of RAM.

TO GET THE CIA ON THE TRAIL OF YOUR DISKS, SEND $65.00 TO:

Send $65.00 Money Order to:
(Checks allow time to clear)

Golden Delicious Software Ltd.
Suite 3308, 350 Fifth Ave.
New York, New York 10001

```
                        2250 *--------------------------------
                        2260 USE.AUXMOVE
                        2270 * Fetch what should be a "0"
                        2280 * or "1" to be AUXMOVE's carry
0836- A4 34             2290          LDY MON.YSAV
0838- B9 00 02          2300          LDA IN,Y
                        2310 * Shift what we fetched to carry
083B- 4A                2320          LSR
                        2330 * Save carry while comparing
083C- 08                2340          PHP
                        2350 * This is a "0" or "1" after a LSR
083D- C9 58             2360          CMP #"0"/2
083F- D0 3E             2370          BNE INVALID.CARRY
0841- E6 34             2380          INC MON.YSAV
                        2390 * Recover Carry
0843- 28                2400          PLP
                        2410 CALL.AUXMOVE.WITH.CARRY
0844- 20 11 C3          2420          JSR AUXMOVE
0847- 60                2430          RTS
                        2440 *--------------------------------
                        2450 USE.XFER
                        2460 * Set XFER Transfer address
                        2470 * from monitor parameter
0848- B5 3C             2480          LDA MON.A1,X
084A- 9D ED 03          2490          STA TRANSFER,X
084D- CA                2500          DEX
084E- 10 F8             2510          BPL USE.XFER
                        2520 * Fetch what should be a "0"
                        2530 * or "1" to be XFER's carry
0850- A4 34             2540          LDY MON.YSAV
0852- B9 00 02          2550          LDA IN,Y
                        2560 * Shift what we fetched to carry
0855- 4A                2570          LSR
                        2580 * Save carry for a while
0856- 08                2590          PHP
                        2600 * This is a "0" or "1" after a LSR
0857- C9 58             2610          CMP #"0"/2
0859- D0 24             2620          BNE INVALID.CARRY
085B- E6 34             2630          INC MON.YSAV
                        2640 * Fetch what should be a "0"
                        2650 * or a "1" to be XFER's overflow
085D- C8                2660          INY
085E- B9 00 02          2670          LDA IN,Y
                        2680 * Shift what we fetched to carry
0861- 4A                2690          LSR
                        2700 * Save this carry too, while we compare
0862- 08                2710          PHP
                        2720 * This is a "0" or "1" after a LSR
0863- C9 58             2730          CMP #"0"/2
0865- D0 17             2740          BNE INVALID.OVERFLOW
0867- E6 34             2750          INC MON.YSAV
                        2760 * Recovered carry is valid overflow
0869- 28                2770          PLP
                        2780 * Move it back to bit 0
086A- 2A                2790          ROL
                        2800 * Recover carry
086B- 28                2810          PLP
                        2820 * Construct overflow
086C- B8                2830          CLV
086D- 29 01             2840          AND #%0000.0001
086F- F0 03             2850          BEQ .1
0871- 2C 7B 08          2860          BIT SEV
                        2870 * Save BSR bank, BSR ram read, and SP
                        2880 * for any calls or returns to main/auxmem
0874- 20 61 09          2890 .1       JSR SAVE.BSR.STATE
0877- BA                2900          TSX
0878- 8E CF 03          2910          STX BANK.SP.SAVE
                        2920 JMP.XFER.WITH.CARRY.AND.OVERFLOW
                        2930 * Routines in aux/main bank may jmp
                        2940 * to RETURN.TO.MAIN/AUX when done
087B- 4C 14 C3          2950 SEV      JMP XFER
                        2960 *--------------------------------
                        2970 INVALID.OVERFLOW
087E- 28                2980          PLP
                        2990 INVALID.CARRY
087F- 28                3000          PLP
                        3010 * Let's not process rest of line
0880- A4 34             3020          LDY MON.YSAV
0882- A9 8D             3030          LDA #$8D
0884- 99 00 02          3040          STA IN,Y
0887- 4C DD FB          3050          JMP BEEP
```

```
                        3060 *--------------------------------
                        3070 SWITCH.MIND
                        3080 * Check in main or aux now
088A- AD 13 C0          3090            LDA READ.RAM.READ.STATUS
088D- 10 03             3100            BPL ENTER.AUX.MON
088F- 4C 25 09          3110            JMP RETURN.TO.MAIN
                        3120 ENTER.AUX.MON
                        3130 * Move USE.AUXMEM to auxmem too
0892- A9 03             3140            LDA #USE.AUXMEM
0894- 85 3C             3150            STA MON.A1
0896- 85 42             3160            STA MON.A4
0898- A9 08             3170            LDA /USE.AUXMEM
089A- 85 3D             3180            STA MON.A1+1
089C- 85 43             3190            STA MON.A4+1
089E- A9 C2             3200            LDA #USE.AUXMEM.END
08A0- 85 3E             3210            STA MON.A2
08A2- A9 09             3220            LDA /USE.AUXMEM.END
08A4- 85 3F             3230            STA MON.A2+1
08A6- 38                3240            SEC
08A7- 20 11 C3          3250            JSR AUXMOVE
                        3260 * Save BSR bank, BSR ram read, and SP
                        3270 * for calls and return to main mem
08AA- 20 61 09          3280            JSR SAVE.BSR.STATE
08AD- BA                3290            TSX
08AE- 8E CF 03          3300            STX BANK.SP.SAVE
                        3310 * Continue in auxmem w/rom
08B1- 8D 03 C0          3320            STA READ.AUX.RAM
08B4- 8D 05 C0          3330            STA WRITE.AUX.RAM
08B7- 8D 09 C0          3340            STA USE.AUX.ZP.STK.BSR
08BA- AD 81 C0          3350            LDA BSR.2.ROM.READ.RAM.WRITE
08BD- AD 81 C0          3360            LDA BSR.2.ROM.READ.RAM.WRITE
                        3370 * What else but this too
08C0- A2 FF             3380            LDX #$FF
08C2- 9A                3390            TXS
                        3400 * Copy rom monitor to auxmem BSR
08C3- A0 00             3410            LDY #MONITOR
08C5- 84 3C             3420            STY MON.A1
08C7- 84 48             3430            STY MON.STATUS
08C9- A9 F8             3440            LDA /MONITOR
08CB- 85 3D             3450            STA MON.A1+1
08CD- B1 3C             3460 .1         LDA (MON.A1),Y
08CF- 91 3C             3470            STA (MON.A1),Y
08D1- C8                3480            INY
08D2- D0 F9             3490            BNE .1
08D4- E6 3D             3500            INC MON.A1+1
08D6- D0 F5             3510            BNE .1
                        3520 * Now use auxmem BSR
08D8- AD 83 C0          3530            LDA BSR.2.RAM.READ.RAM.WRITE
08DB- AD 83 C0          3540            LDA BSR.2.RAM.READ.RAM.WRITE
                        3550 * Fix monitor in BSR
08DE- A9 09             3560            LDA /DO.CLREOL
08E0- 8D 8D FD          3570            STA MON.JSR.CLREOL+2
08E3- A9 6E             3580            LDA #DO.CLREOL
08E5- 8D 8C FD          3590            STA MON.JSR.CLREOL+1
                        3600 * Hook I/O through main
08E8- A9 70             3610            LDA #COUT.TO.MAIN
08EA- 85 36             3620            STA MON.CSWL
08EC- A9 90             3630            LDA #RDKEY.FROM.MAIN
08EE- 85 38             3640            STA MON.KSWL
08F0- A9 09             3650            LDA /COUT.TO.MAIN
08F2- 85 37             3660            STA MON.CSWL+1
                        3670 *          LDA /RDKEY.FROM.MAIN
08F4- 85 39             3680            STA MON.KSWL+1
                        3690 * USE.AUXMEM in auxmem too
08F6- 20 12 08          3700            JSR CONNECT.CONTROL.Y
                        3710 * Do page 3 loc
08F9- 8D FB 03          3720            STA NMI
08FC- A9 65             3730            LDA #MON
08FE- 8D FC 03          3740            STA NMI+1
0901- 8D FE 03          3750            STA MON.IRQLOC
0904- A9 FF             3760            LDA /MON
0906- 8D FD 03          3770            STA NMI+2
0909- 8D FF 03          3780            STA MON.IRQLOC+1
090C- A9 59             3790            LDA #MON.OLDBRK
090E- 8D F0 03          3800            STA MON.BRKV
0911- A9 FA             3810            LDA /MON.OLDBRK
0913- 8D F1 03          3820            STA MON.BRKV+1
                        3830 * Enter monitor in auxmem BSR
0916- 4C 65 FF          3840            JMP MON
```

D O W N L O A D I N G
CUSTOM CHARACTER SETS

One of the features 'hidden' in many printers available today is their ability to accept user-defined character sets. With the proper software, these **custom characters** are 'downloaded' from your Apple II computer to the printer in a fraction of a second. Once the printer has 'learned' these new characters, they will be remembered until the printer is turned off.

After the downloading operation, you can use your printer with virtually any word processor. Just think of the possibilities! There's nothing like having your own **CUSTOM CHARACTERS** to help convey the message. And you still have access to those built-in fonts as well! **Here's a quick look at some possible variations:**

	BUILT-IN	CUSTOM
10CPI:	AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
12CPI:	AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
17CPI:	AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
5CPI:	AaBbCcDdEeFf	**AaBbCcDdEeFf**
6CPI:	AaBbCcDdEeFf	**AaBbCcDdEeFf**
8CPI:	AaBbCcDdEeFf	**AaBbCcDdEeFf**

And let's not forget Enhanced and Underined printing as well...

AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**

The Font Downloader & Character Editor software package has been developed by RAK-WARE to help you unleash the power of your printer. The basic package includes the downloading software with 4 fonts to get you going. Also included is a character editor so that you can turn your creativity loose. Use it to generate unique character fonts, patterns, symbols and graphics. A detailed user's guide is provided on the program diskette.

SYSTEM REQUIREMENTS:
* APPLE II, APPLE II Plus, APPLE //e or lookalike with 48K RAM
* 'DUMB' Parallel Printer Interface Board (like Apple's Parallel Printer Interface, TYMAC's PPC-100 or equivalent)

The Font Downloader & Editor package is only $39.95 and is currently available for either the Apple Dot Matrix Printer or C.Itoh 8510AP (specify printer). Epson FX-80 and OkiData versions coming soon. Enclose payment with order to avoid $3.00 handling & postage charge.

R A K - W A R E
41 Ralph Road West Orange New Jersey 07052

Say You Saw It In **APPLE ASSEMBLY LINE**!

```
                        3850 *--------------------------------
                        3860 RETURN.TO.AUX
                        3870 * Continue in aux ram
0919- 8D 03 C0          3880          STA READ.AUX.RAM
091C- 8D 05 C0          3890          STA WRITE.AUX.RAM
091F- 8D 09 C0          3900          STA USE.AUX.ZP.STK.BSR
0922- 4C 2E 09          3910          JMP RETURN.COMMON
                        3920 RETURN.TO.MAIN
                        3930 * Continue in main ram
0925- 8D 02 C0          3940          STA READ.MAIN.RAM
0928- 8D 04 C0          3950          STA WRITE.MAIN.RAM
092B- 8D 08 C0          3960          STA USE.MAIN.ZP.STK.BSR
                        3970 RETURN.COMMON
                        3980 * Recover SP
092E- AE CF 03          3990          LDX BANK.SP.SAVE
0931- 9A                4000          TXS
                        4010 RESTORE.BSR.STATE
0932- B8                4020          CLV
0933- AE CD 03          4030          LDX BANK.BSR.BANK.SAVE
0936- 10 15             4040          BPL .2
0938- AE CE 03          4050          LDX BANK.BSR.RAM.READ.SAVE
093B- 10 08             4060          BPL .1
093D- AE 83 C0          4070          LDX BSR.2.RAM.READ.RAM.WRITE
0940- AE 83 C0          4080          LDX BSR.2.RAM.READ.RAM.WRITE
0943- 50 1B             4090          BVC .4
0945- AE 81 C0          4100 .1       LDX BSR.2.ROM.READ.RAM.WRITE
0948- AE 81 C0          4110          LDX BSR.2.ROM.READ.RAM.WRITE
094B- 50 13             4120          BVC .4
094D- AE CE 03          4130 .2       LDX BANK.BSR.RAM.READ.SAVE
0950- 10 08             4140          BPL .3
0952- AE 8B C0          4150          LDX BSR.1.RAM.READ.RAM.WRITE
0955- AE 8B C0          4160          LDX BSR.1.RAM.READ.RAM.WRITE
0958- 50 06             4170          BVC .4
095A- AE 89 C0          4180 .3       LDX BSR.1.ROM.READ.RAM.WRITE
095D- AE 89 C0          4190          LDX BSR.1.ROM.READ.RAM.WRITE
0960- 60                4200 .4       RTS
                        4210 *--------------------------------
                        4220 SAVE.BSR.STATE
0961- AE 11 C0          4230          LDX READ.BSR.BANK
0964- 8E CD 03          4240          STX BANK.BSR.BANK.SAVE
0967- AE 12 C0          4250          LDX READ.BSR.RAM.READ
096A- 8E CE 03          4260          STX BANK.BSR.RAM.READ.SAVE
096D- 60                4270          RTS
                        4280 *--------------------------------
                        4290 DO.CLREOL
096E- A9 9D             4300          LDA #"]"-'@'
                        4310 COUT.TO.MAIN
                        4320 * Save auxmem's X
0970- 8E CC 03          4330          STX BANK.X.SAVE
                        4340 * Save BSR bank, BSR ram read, and SP
                        4350 * over call to main ram
0973- 20 61 09          4360          JSR SAVE.BSR.STATE
0976- BA                4370          TSX
0977- 8E CF 03          4380          STX BANK.SP.SAVE
                        4390 * Continue in main ram
097A- 8D 02 C0          4400          STA READ.MAIN.RAM
097D- 8D 04 C0          4410          STA WRITE.MAIN.RAM
0980- 8D 08 C0          4420          STA USE.MAIN.ZP.STK.BSR
                        4430 * Recover SP
0983- AE CF 03          4440          LDX BANK.SP.SAVE
0986- 9A                4450          TXS
0987- 20 32 09          4460          JSR RESTORE.BSR.STATE
098A- 20 ED FD          4470          JSR MON.COUT
098D- 4C AF 09          4480          JMP IO.COMMON
                        4490 *--------------------------------
                        4500 RDKEY.FROM.MAIN
                        4510 * Repair monitor's sillier attempt
0990- 91 28             4520          STA (MON.BASL),Y
                        4530 * Save auxmem's X
0992- 8E CC 03          4540          STX BANK.X.SAVE
                        4550 * Save BSR bank, BSR ram read, and SP
                        4560 * over call to main ram
0995- 20 61 09          4570          JSR SAVE.BSR.STATE
0998- BA                4580          TSX
0999- 8E CF 03          4590          STX BANK.SP.SAVE
```

```
                    4600 * Continue in main ram
099C- 8D 02 C0      4610            STA READ.MAIN.RAM
099F- 8D 04 C0      4620            STA WRITE.MAIN.RAM
09A2- 8D 08 C0      4630            STA USE.MAIN.ZP.STK.BSR
09A5- AE CF 03      4640            LDX BANK.SP.SAVE        Recover SP
09A8- 9A            4650            TXS
09A9- 20 32 09      4660            JSR RESTORE.BSR.STATE
09AC- 20 0C FD      4670            JSR MON.RDKEY
                    4680 *-------------------------------
                    4690 IO.COMMON
09AF- 8D 03 C0      4700            STA READ.AUX.RAM        Continue in Aux RAM
09B2- 8D 05 C0      4710            STA WRITE.AUX.RAM
09B5- 8D 09 C0      4720            STA USE.AUX.ZP.STK.BSR
09B8- AE CF 03      4730            LDX BANK.SP.SAVE        Recover SP
09BB- 9A            4740            TXS
09BC- 20 32 09      4750            JSR RESTORE.BSR.STATE
09BF- AE CC 03      4760            LDX BANK.X.SAVE         Recover X
09C2- 60            4770            RTS
```

65C02

People who have started reading AAL since last December have asked what is all this 65C02 business, anyway? Well the 65C02 is a new CMOS version of the 6502 microprocessor. (CMOS stands for Complementary Metal Oxide Semiconductor. That's a different way of making chips. CMOS circuits are noted for extremely low power consumption and extremely high sensitivity to static electricity.) To us Apple owners, the important thing is that the designers of the new chip corrected the bugs in the 6502 and added several new instructions and addressing modes.

The new instructions include PHX, PLX, PHY, and PLY (push and pull the X and Y registers from the stack), BRA (branch always), STZ (store zero), TSB and TRB (test and set or reset bits), and SMB, RMB, BBR and BBS (set, reset and branch on single bits). The main new addressing mode is true indirect without indexing, LDA ($12). This mode is now available for ORA, AND, EOR, ADC, STA, LDA, CMP, and SBC. There are also new modes for the BIT and JMP instructions. INC and DEC can now work on the A register.

There are some problems, though. Rockwell, GTE, NCR, and Synertek (maybe) are manufacturing 65C02 processors, but they are not all the same. The SMB, RMB, BBS, and BBR instructions are only available in the Rockwell chip. The NCR chip works in the Apple //e, but not in older Apples. The GTE processor does work in all Apples (this is being written on an Apple][+ with a GTE 65C02). I haven't yet gotten a sample of the Rockwell processor, so I don't personally know if it works in older Apples. Some people say yes, others no.

That's a summary of what we know so far. The confusion is beginning to clear up, but there are still questions about what will or won't work in which Apples, and why. Stay tuned...

Speeding Up Spirals.....Bob Sander-Cederlof

Several have written to us about Roger Keating's Spiral Screen Clear (AAL June 1983). Charles Putney, who you may remember as the first one to double the speed of the prime number program in AAL several years ago, has now applied his talent to unwinding the screen.

Roger's program ran in 55 seconds, my table-lookup for BASCALC shortened it to 40 seconds. Charlie wrote the whole thing out as one long string of LDA-STA pairs, and trimmed the time to only 7 seconds!

Let's see...there are 960 characters on the screen. If I write a LDA-STA pair to move each byte ahead one position along the spiral path, I will have 959 such pairs. Each LDA and each STA will take 3 bytes, so the program to shift the whole screen one step around the spiral path will take 2x3x959 = 5754 bytes. Add another 5 bytes to LDA #$A0 and store it in the center of the screen before the first rotation. Then add some code to re-run the 959 steps 959 more times, so that the whole screen clears, and you get Charlie's program, 5777 bytes.

Now try to type it all in! Don't worry, we aren't even going to list it here. It will be on the next Quarterly disk, though.

Charlie decided to use five macros, to decrease the amount of manual labor involved. He defined a macro named MOVE which builds the LDA-STA pair for a pair of arguments:

```
.MA MOVE
LDA ]1
STA ]2
.EM
```

Then he defined one macro for each leg of the spiral: MOVED, MOVEL, MOVEU, and MOVER for down, left, up, and right respectively. With a few comment lines, the macro definitions take a mere 488 lines! The macros are each called with three parameters:

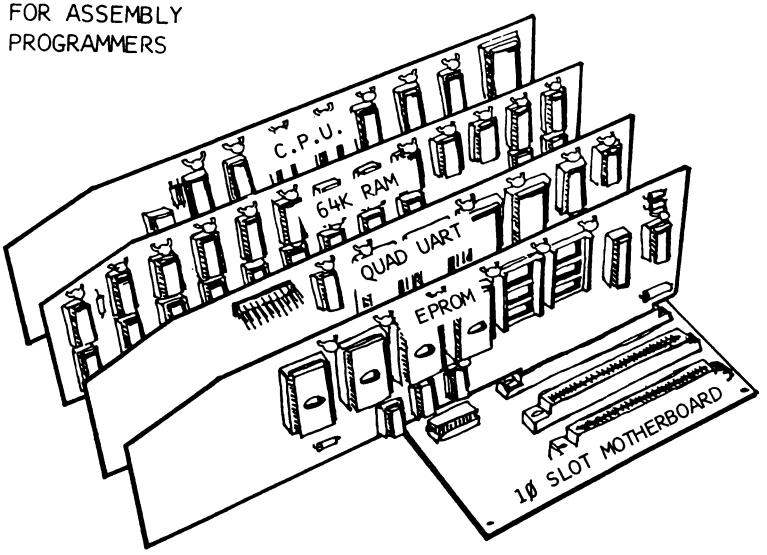
```
>MOVED col,low.row,high.row
>MOVEL row,low.col,high.col
>MOVEU col,low.row,high.row
>MOVER row,low.col,high.col
```

The definitions out of the way, it only remains to write 12 sets of 4 macro calls, or 48 lines, and a driving loop to do it all 960 times. Here is a condensed listing of the actual code part of Charlie's program:

```
6400 *--------------------------------
6410 *
6420 *       SPIRAL PROGRAM
6430         .OR $6000      OUT OF THE WAY
6440         .TF SPIRAL.OBJ
6450 *
6460 *
6470 SPIRAL LDA #' '+$80 GET A SPACE
6480         STA R12+12    PUT IT IN CENTER
6490         LDX #960      HOW MANY TIMES ?
6500         LDY /960      HIGH ORDER
6510 *
6520 SPI1    >MOVED 0,0,23
6530         >MOVEL R0,0,39
6540         >MOVEU 39,0,23
6550         >MOVER R23,1,39
6560 *
6570         >MOVED 1,1,23
6580         >MOVEL R1,1,38
6590         >MOVEU 38,1,22
6600         >MOVER R22,2,38
6610 *
6620         >MOVED 2,2,22
6630         >MOVEL R2,2,37
6640         >MOVEU 37,2,21
6650         >MOVER R21,3,37
6660 *
6670         >MOVED 3,3,21
6680         >MOVEL R3,3,36
6690         >MOVEU 36,3,20
6700         >MOVER R20,4,36
6710 *
6720         >MOVED 4,4,20
6730         >MOVEL R4,4,35
6740         >MOVEU 35,4,19
6750         >MOVER R19,5,35
6760 *
6770         >MOVED 5,5,19
6780         >MOVEL R5,5,34
6790         >MOVEU 34,5,18
6800         >MOVER R18,6,34
6810 *
6820         >MOVED 6,6,18
6830         >MOVEL R6,6,33
6840         >MOVEU 33,6,17
6850         >MOVER R17,7,33
6860 *
6870         >MOVED 7,7,17
6880         >MOVEL R7,7,32
6890         >MOVEU 32,7,16
6900         >MOVER R16,8,32
6910 *
6920         >MOVED 8,8,16
6930         >MOVEL R8,8,31
6940         >MOVEU 31,8,15
6950         >MOVER R15,9,31
6960 *
6970         >MOVED 9,9,15
6980         >MOVEL R9,9,30
6990         >MOVEU 30,9,14
7000         >MOVER R14,10,30
7010 *
7020         >MOVED 10,10,14
7030         >MOVEL R10,10,29
7040         >MOVEU 29,10,13
7050         >MOVER R13,11,29
7060 *
7070         >MOVED 11,11,13
7080         >MOVEL R11,11,28
7090         >MOVEU 28,11,12
7100         >MOVER R12,12,28
7110 *
7120         DEX
7130         CPX #$FF
7140         BNE SPI2
7150         DEY
7160         CPY #$FF
7170         BNE SPI2
7180         RTS
7190 SPI2    JMP SPI1
```

APPLESEED

HARDWARE FOR ASSEMBLY
LANGUAGE PROGRAMMERS



Appleseed boards allow systems to be designed and programmed on a standard Apple Computer and then produced with a board set which costs a fraction of what an Apple Computer would cost. This approach allows the use of a multitude of sophisticated programming and hardware tools which have been designed for use on the Apple Computer. Low cost, small size, low power consumption, and flexibility are the advantages with **Appleseed.** All of the Apple bus conventions are preserved which allows the use of almost any custom board which has been designed for use in the Apple computer. Programs are developed using a good editor/assembler (such as S-C Macro); then after the system is debugged and running properly, a set of EPROMS are blown and inserted into the destination machine. Soon we will have available a set of boards which will allow direct loading of the program from your Apple computer into the **Appleseed** for the testing of the program prior to actual blowing of a set of EPROMS.

The **Appleseed** approach allows the greatest possible flexibility in system design. You can pick and choose among the various **Appleseed** components as well as a multitude of peripheral boards made by others in order to design your own customized system. The only limitation is that we don't supply firmware and we don't support the standard Apple CRT graphics. Flexibility is the keyword! You buy only what you need for your application.

Optional packaging is also available. We have a "cute little box with walnut side panels" or a 19 inch rack mount with built-in disk drives.

Appleseed products are not available through computer stores. For more information call or write us directly. Also look for us at the Applefest in San Francisco, October 28th-30th.

Apple is a registered trademark of Apple Computer Inc.

DOUGLAS ELECTRONICS - 718 MARINA - SAN LEANDRO, CA 94577 - (415)483-8770

Remember, the whole source with the full macro definitions will be on the next quarterly disk ($15, for all source code in issues July-August-September 1983).

Because Charlie's program makes such heavy use of macros, it takes considerable time to assemble. He timed it at nearly two minutes. If the program were written out the long way, without macros, it would take only about 20 seconds to assemble.

Charlie pointed out that we are needlessly moving the center of the spiral, which is already blank. As the blanked portion grows, this becomes very significant. In fact, by eliminating moving the cleared portion, the time could be further reduced to only 3 1/2 seconds. Each LDA-STA takes 8 cycles. The long way takes 959*960 pairs, plus some overhead. Ignoring the overhead, we get 7365120 cycles, or about 7.2 seconds. Forgetting the blanked stuff makes it 3.6 seconds. Any takers?

And I was just wondering...how about an Applesoft program which writes the 959 LDA-STA pairs as assembly language source on a text file? Or POKEs the actual object code, by computing the addresses necessary, into a binary buffer area. Again, any takers?

Tinkering with Variable Cross Reference.....Louis Pitz
De Witt, Iowa

I am a tinkerer! Yes I love to take programs and add features to improve them. Sometimes the "improved" version even works! Usually I learn a lot about humility, and occasionally a bit about programming.

A case in point is the program for doing an Applesoft Variable Cross Reference (from the November 1980 issue of Apple Assembly Line). I just recently got Quarterly Disk #1 with its source code, and so it became "tinker-time".

VCR works just fine, and is fast! But it only produces 40-column output, and I wanted both 40-column screen and 80-column printer hardcopy. Here are some patches which will do the job. It makes a good short example of changing output hooks in the middle of a program without goofing up DOS.

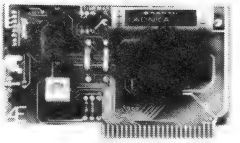
```
1060        .TF B.VCRP   "P" FOR PRINTER VERSION

4534         LDA #0      RESET COUNTER TO 0
4538         STA $6      FOR EACH VARIABLE

4821         INC $6      COUNT THE SCREEN LINE
4822         LDA $6
4823         AND #1      LOOK AT ODD-EVEN BIT
4824         BEQ TAB.NEW.LINE   BOTH SCRN AND PRINTER
4825         LDA #$FDF0         ONLY SCRN GETS NEW LINE
4826         STA $36     SO DISCONNECT PRINTER
4827         LDA /$FDF0
4828         STA $37
4829         JSR $3EA           PASS TO DOS
4830         JSR MON.CROUT      SCREEN ONLY
4831         LDA #$C100         REHOOK PRINTER
4832         STA $36
4833         LDA /$C100
4834         STA $37
4835         JSR $3EA           PASS TO DOS
4836         BNE .1             ...ALWAYS
```

To use the printer version of VCR, BRUN B.VCRP. This sets up the ampersand vector. Then LOAD your Applesoft program. Use PR#1 to turn on your printer. Then type "&" and RETURN, and watch the cross reference.

If your printer is in some slot other than 1, change lines 4831 and 4833 to the correct value ($Cs00, where s=slot#).

APPLIED ENGINEERING
THE BEST PERIPHERALS FOR THE BEST COMPUTER

The TIMEMASTER
Finally a clock that does it ALL!

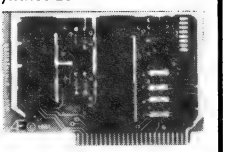


- Designed in 1983 using I.C. technologies that simply did not exist when most other Apple clocks were designed.
- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- Powerful 2K ROM driver — No clock could be easier to use.
- Full emulation of most other clocks, including Mountain Hardware's Appleclock (but you'll like the TIMEMASTER mode better).
- Basic, Machine Code, CP/M and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time. (Many examples are included)
- On board timer lets you time any interval up to 48 days long down to the nearest millisecond.

The TIMEMASTER includes 2 disks with some really fantastic time oriented programs (over 25) plus a DOS dater so it will automatically add the date when disk files are created or modified. This disk is over a $200.00 value alone — we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER.

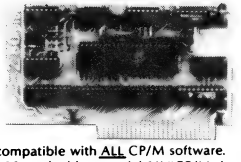
If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER. **PRICE $129.00**

Super Music Synthesizer



- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all the features of this board. Their software sounds the same in our synthesizer.)
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features. **PRICE $159.00**

Z-80 PLUS!



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- All new 1983 design incorporates the latest in I.C. technologies.
- Red "CP/M WORKING" LED indicator, the Z-80 Plus does not interfere with non-CP/M programs.
- An on-card PROM eliminates many I.C.'s for a cooler, less power consuming board. (We use the Z-80A at a fast 4MHZ)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE $139.00

COMING SOON: The Z-80 Plus for the Apple III

Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

- TOTALLY Videx Compatible
- 80 characters by 24 lines, with a sharp 7x9 dot matrix
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with ALL Apple languages and software — there are NO exceptions
- Low power consumption through the use of CMOS devices
- All connections on the card are made with standard video connectors, no cables are soldered to the board
- All new 1983 design (using a new Microprocessor based C.R.T. controller)

JUST COMPARE!

VIEWMASTER	**169**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
SUPRTERM	375	NO	YES	NO	NO	NO	YES	YES
WIZARD80	245	NO	NO	YES	YES	NO	YES	YES
VISION80	375	YES	YES	YES	YES	NO	NO	NO
OMNIVISION	295	NO	YES	NO	NO	NO	YES	YES
VIEWMAX80	219	YES	YES	YES	YES	NO	NO	YES
SMARTERM	360	YES	YES	YES	NO	NO	YES	NO
VIDEOTERM	345	NO	NO	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, Viscalc, and many others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE $169.00

MemoryMaster IIe

- Expands your Apple IIe to 192K memory
- Provides an 80 column text display
- Compatible with all Apple IIe 80 column and extended 80 column card software (Same physical size as Apple's 64K card)
- Available in 64K and 128K configurations
- Bank select LED's for each 64K bank
- Permits your IIe to use the new double high resolution graphics
- Automatically expands Visicalc to 95K storage in 80 columns! The 64K configuration is all that's needed, 128K can take you even higher.

128K RAM Card

- Complete documentation included, we show you how to use all 128K.

If you already have Apple's 64K card, just order the MEMORYMASTER with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster with 128K	**$249**
Upgradeable MemoryMaster with 64K	**$169**
Non-Upgradeable MemoryMaster with 64K	**$149**

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in APPLE IIe, II, II+ and Franklin (except MemoryMaster). Applied Engineering also manufactures a full line of data acquisition and control products for the Apple; A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

All Orders Shipped Same Day. Texas Residents Add 5% Sales Tax. Add $10.00 If Outside U.S.A. Dealer Inquiries Welcome.

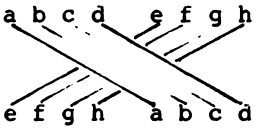
Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 470301
Dallas, TX 75247

Call (214) 492-2027
7a.m. to 11p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome

Reversing, Getting, and Putting Nybbles....Bob Sander-Cederlof

In the process of de-crypting a large data base, I needed to reverse the nybbles in each of roughly 32000 bytes. There are probably a lot of ways to do this, but I found one which takes only 12 bytes to reverse the nybbles in the A-register.

Just to be sure we agree on what I am talking about, here is a little diagram:



One way, sort of brute force, involves breaking the nybbles out and remerging them:

```
        LDA (PNTR),Y
        ASL             SHIFT EFGH LEFT
        ASL
        ASL
        ASL
        STA TEMP
        LDA (PNTR),Y
        LSR             SHIFT ABCD RIGHT
        LSR
        LSR
        LSR
        ORA TEMP        RE-MERGE NYBBLES
        STA (PNTR),Y
```

From another perspective, I am trying to rotate the data byte half-way around. But if I try to do it with ROL or ROR instructions, one bit gets left in CARRY, and an extra bit gets inserted in the middle. Here is how I finally did it:

```
        LDA (PNTR),Y    abcd efgh
        ASL             bcde fgh0
        ADC #0          bcde fgha
        ASL             cdef gha0
        ADC #0          cdef ghab
        ASL             defg hab0
        ADC #0          defg habc
        ASL             efgh abc0
        ADC #0          efgh abcd
        STA (PNTR),Y
```

Each ASL-ADC pair shifts the byte around one bit. The ASL shifts the leftmost bit into the CARRY bit, and a zero into the right end. The ADC #0 adds CARRY into the rightmost bit.

Naturally, curiosity forces me to look at the possibility of shifting right one bit also. We have LSR and ROR, of course, but both of these leave the shifted out bit in CARRY. I want that bit back in the sign position, like this:

ABCDEFGH should become HABCDEFG

Two similar methods come to mind, depending on how I might use it. If the byte to be shifted is in A-reg, and needs to remain there, and I don't want to upset any other registers, I can do it like this:

```
PHA     save unshifted value
LSR     get rightmost bit in CARRY
PLA     restore unshifted value
ROR     shift again, putting right bit on left
```

If the byte to be shifted is in memory, and I want the results to be in memory, I might do it like this:

```
LDA FLAG
LSR         RIGHTMOST BIT INTO CARRY
ROR FLAG    SHIFT BYTE, PUTTING RIGHT INTO LEFT
```

Note that I can branch according to the value of the bit which moved around by using BMI or BPL, because that bit is the new sign bit.

The last method above can be useful when you have a program that needs to alternate between two paths. For example, suppose I write a program to pick up the "next nybble" from a data area. The first time I call it, I want to get the left nybble of the first byte. Next time, the right nybble of the same byte. Next time the left nybble of the next byte. And so on.

I might store the value $55 in FLAG initially, and then use LDA FLAG, LSR, ROR FLAG, to shift it around. FLAG will alternate between $55 and $AA. My subroutine can alternate between left and right nybbles.

Not to leave you hanging, I wrote "get next nybble" and "put next nybble" subroutines. By the time I finished polishing, yet another technique had surfaced for rotating the $55/$AA flag. I used this new method so as not disturb the contents of the A-register.

To set up either routine, the address of the beginning of the data area must be put into PNTR and PNTR+1, and $55 must be put into FLAG.

```
                    1000 *SAVE NYBBLE GET & PUT
                    1010 *-------------------------------
0000-               1020 PNTR   .EQ 0 AND 1
0002-               1030 FLAG   .EQ 2
                    1040 *-------------------------------
                    1050 *      PUT NEXT NYBBLE AT (PNTR)
                    1060 *      IF FLAG = $55, PUT LEFT NYBBLE
                    1070 *              = $AA, PUT RIGHT NYBBLE
                    1080 *-------------------------------
                    1090 PUT.NEXT.NYBBLE
0800- A2 00         1100        LDX #0
0802- 46 02         1110        LSR FLAG            $55 OR $AA
0804- B0 07         1120        BCS .1              ...IT WAS $AA, NOW $54
                    1130 *---STORE IN LEFT NYBBLE---------
0806- 0A            1140        ASL          FLAG NOW $AA
0807- 0A            1150        ASL
0808- 0A            1160        ASL
0809- 0A            1170        ASL
080A- 81 00         1180        STA (PNTR,X)
080C- 60            1190        RTS
```

```
                    1200 *---STORE IN RIGHT NYBBLE--------
080D- 01 00         1210 .1     ORA (PNTR,X) MERGE WITH LEFT NYBBLE
080F- 81 00         1220        STA (PNTR,X)
0811- E6 02         1230        INC FLAG     MAKE $54 INTO $55
0813- E6 00         1240        INC PNTR     MOVE PNTR TO NEXT BYTE
0815- D0 02         1250        BNE .2
0817- E6 01         1260        INC PNTR+1
0819- 60            1270 .2     RTS
                    1280 *--------------------------------
                    1290 *      GET NEXT NYBBLE
                    1300 *      IF FLAG = $55, GET LEFT NYBBLE
                    1310 *              = $AA, GET RIGHT NYBBLE
                    1320 *--------------------------------
                    1330 GET.NEXT.NYBBLE
081A- A2 00         1340        LDX #0
081C- 46 02         1350        LSR FLAG          WAS $55 OR $AA
081E- A1 00         1360        LDA (PNTR,X)    GET BYTE WITH NYBBLES
0820- B0 05         1370        BCS .1            ...WAS $AA, NOW $54
                    1380 *---GET LEFT NYBBLE--------------
0822- 4A            1390        LSR
0823- 4A            1400        LSR
0824- 4A            1410        LSR
0825- 4A            1420        LSR
0826- 60            1430        RTS
                    1440 *---GET RIGHT NYBBLE--------------
0827- E6 02         1450 .1     INC FLAG     MAKE $54 INTO $55
0829- E6 00         1460        INC PNTR     ADVANCE TO NEXT BYTE
082B- D0 02         1470        BNE .2
082D- E6 01         1480        INC PNTR+1
082F- 29 0F         1490 .2     AND #$0F     ISOLATE NYBBLE
0831- 60            1500        RTS
                    1510 *--------------------------------
```

Grappler Interfaces

There should be a leaflet included with this issue describing the Grappler printer interfaces. We now have three of them "in the family" here, and have been very pleased with their performance. Check the brochure for features, the ad on page three for our prices, and let us hear from you.

WICO Track Ball

Several of you have inquired about or ordered the WICO Track Ball that I reviewed a couple of months ago, so we've decided to carry them regularly. WICO has since raised their price from $79.95 to $89.95, so we're going from $75 to $80.

Diskettes

There's getting to be a lot more competition in the diskette business, so prices are falling. After seeing so many ads at such attractive prices, Bob called Verbatim and told them that we had to have a better price, or we would have to change brands. That paid off, so we can now offer the same high-quality Verbatim Datalife diskettes at $45.00 for a package of 20. That's $2.25 each for the best diskettes we've found.

Whatever You Want

If you're shopping for a new peripheral, accessory, or program, give us a call and ask for a quote. We can get nearly anything you might want, and we'd love the chance to serve you.

Some Small Patches.....Bill Morgan

We've had several calls requesting the patch addresses for a couple of features in the S-C Macro Assemblers.

Ansert?

In Version 1.1 of the Macro Assembler, Bob changed the CTRL-I (Insert) command in the EDIT mode to CTRL-A (for ADD). This was done because the Apple //e keyboard has the TAB key, which generates a CTRL-I code. It didn't seem to make much sense to have the TAB key do an insert operation, so he added a clear-to-next-tab-stop function for CTRL-I.

Well, a lot of people don't have //e's, or don't much care about the TAB key. A lot of us are used to CTRL-I for Insert, and would like to keep it that way.

The CTRL-A character ($81) is at $1C87 in the $1000 version, and at $DCB7 in the $D000 version. Just change that byte to a $89, and you'll have your good old CTRL-I back. If you want to keep the clear-to-tab-stop function, you can change the $89 at $1CC6 ($DCC6) to a $81. That will make CTRL-A do the clear-to-tab.

.BS Filler Byte

The directive .BS <expr> skips over <expr> bytes when you are assembling to memory, and sends <expr> zero bytes to the target file when you are assembling to disk. Several people have asked how to change the zero to some other value.

For example, a freshly-erased EPROM contains all $FF bytes. When you burn data into the chip, you actually write in just the zero bits. If you are assembling code to be written into an EPROM, you want any fill bytes to be $FF, so you can add patches later without having to erase and re-write the whole chip.

The following table shows the addresses of the zero byte in the various versions of the Macro Assembler. Just change the indicated byte to the value you want to use for filler.

Version	1.0	1.1			
		40-col	//e	Videx	STB
$1000	2D43	2D62	2D48	2E37	2E60
$D000	EE8F	EE86	EE62	EF5A	EF83

Some More 68000 Boards

First let me apologize for an erroneous statement in the May '83 issue, in which I juxtaposed two unrelated facts in a cause-effect sentence. Many readers have sent corrections: I am told that grounding the DTACK signal has nothing to do with how much memory you can add. How did I ever get the idea that it did? If you want the straight scoop on this, subscribe to Digital Acoustics' newsletter "DTACK Grounded".

Digital Acoustics has announced a new board, called the "DTACK Grande". Almost sounds like "grounded", but this time it isn't. You get one megabyte of RAM and a 12.5 MHz 68000. RAM refresh is handled by an interrupt routine, with software. The overhead is only 4%, giving an effective speed of 10 MHz. Expansion connectors on the card can connect to another 15.7 megabytes. I'd say Saybrook has been passed by, but Hal Hardenburg beat me to it! (Digital Acoustics, 1415 E. McFadden, Suite F, Santa Ana, CA 92705. (714) 835-4884)

Mike Heckman at Anthro-Digital sent me some literature on another new 68000 board. Enhancement Technology Corporation calls it the "PDQ//". Specs include: 10 MHZ, 256K RAM, UCSD p-system, Applesoft-compatible BASIC. The price will be $1495, available by the end of August. We may be able to make you a deal on one of these. (ETC, P.O.Box 1267, Pittsfield, MA 01202. (413) 445-4219)

Bringing Some Patches Together.....Jim Wetzel

Earlier this year I decided to break down and finally buy an 80-column card for my Apple II+. After all, it's cheaper than a IIe. I was just about to type in the Videx patches from AAL Volume 2, No.11, when Bob announced Version 1.1. Well with the Videx patches and all the new features I just couldn't pass up his offer. After a call to Bob and a three day wait I had version 1.1 of the S-C Macro Assembler.

While testing out the new version I soon discovered most of the patches I had applied to version 1.0 would not work properly. The addresses of the routines/tables had all moved. After a few hours work and a lot of dis-assembling I would like to share the new locations with AAL readers and bring some of the patches together.

First I will describe the new addresses and then show how I used them.

The Escape Function Table is now located at $14AB-$14C6 <ESC-@ thru ESC-M>. This is a group of two-byte addresses (minus 1, because they are of the PHA-PHA-RTS variety) of the routines to handle the escape functions.

The Edit Function Table is now located at $1CB4-$1CE3 <ctrl-@ thru ctrl-X>. This table is somewhat different. Each entry is three bytes long and it contains the control character and the address-minus-1 of the routine to handle the function.

Location $14D3 contains the dash count <$26> for the ESC-L function.

Location $13FF contains a JSR to the monitor Bell routine. This is the end of the input checker, the JSR BELL is executed when an invalid character is entered, and a good place to put a JSR to an extended input processor.

These locations are valid for the regular version and the Videx version which load at $1000. For language card users just add $C000 to the address. My hat is off to Bob for adding all the features of Videx and still keeping the assembler looking the same. I have not checked the STB or the IIe versions for compatibility but, with a little bit of work and knowing what to look for it should be an easy process.

Now, what can you do with this information? I have modified Bob's language card loader to show you (figure 1). With the exception of the REM statements, lines 1000-1140 of the file are as Bob supplied; after that the changes begin. I will not spend a lot of time explaining the routines themselves because they are all well documented in the referenced AAL articles.

The first thing I do is load in my extended input processor (figure 2) at $F600. There appears to be about two free pages after the assembler and before monitor in the language card version. For standard version users just move the symbol table up as described AAL Vol. 2, No. 9. My input processor is a

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.

Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

combination of Auto Catalog (AAL 2.9) and Toggling Upper/Lower Case (AAL 3.3). Next I modify the JSR BELL to JSR CONTROL.A.

Once you have control you can add any routines you wish (such as R. F. O'Brien's Auto/Manual Toggle AAL 2.11). For now I am only interested in an upper/lower case toggle.

Line 1170 modifies the ESC-C function to JSR to my routine for auto Catalog. Remember this should be the address of the routine - 1. Lines 1180-1190 change the cursor to a blinking underline (as described in AAL 3.5) along with line 1200 which changes the number of "-"'s from 38 to 64 (I found 68 to be too many).

Last but not least is an answer to Steve Mann's request for a upper/lower case toggle in EDIT mode. In version 1.1 Bob changed the ctrl-I key function in EDIT mode and added a ctrl-A key function in its place. He did it so that the //e TAB key, which generates control-I, would really mean TAB.

Well Bob, I like mnemonic commands (like ctrl-I for Insert), and think the older Apples should still take precedence. Line 1210 changes the ctrl-A key to branch to my upper/lower case toggle routine, just past the character check, and line 1220 changes the ctrl-I routine back to its proper function (this was the address found in the ctrl-A area).

I hope these patches will be useful to other AAL readers not only for what they do, but for how they do it.

```
1000 REM LOAD S-C MACRO ASSEMBLER (VIDEX)
1010 REM INTO RAM AT $D000
1020 REM LOAD PATCHES AT $F600
1030 REM PATCH INPUT TEST TO CHECK FOR MY COMMANDS BEFORE ERROR
1040 REM PATCH ESCAPE TABLE ($D4AB-) FOR ESCAPE-C
1050 REM CHANGE CURSOR TO BLINKING UNDERLINE
1060 REM PATCH ESC-L DASH LINE COUNT
1070 REM PATCH EDIT CNTL-A TO MY ROUTINE
1080 REM PATCH EDIT CNTL-I BACK TO INSERT FUNCTION
1090 CALL-151
1100 C081 C081
1110 F800<F800.FFFFM
1120 BLOAD S-C.ASM.MACRO.D000.VIDEX
1130 300:A9 4C CD 00 E0 F0 12 8D 00 E0 A9 00 8D 01 E0 A9 D0
1140 300G
1150 BLOAD SCM.PATCH              8D 02 E0 A9 CB 8D D1 03 60
1160 D3FF:20 00 F6
1170 D4B1:19 F6
1180 C0B0:0A 68
1190 C0B0:0B 08
1200 D4D3:40
1210 DCB8:03 F6
1220 DCC7:0B DC
1230 C080
1240 3D3G
```

S-C Macro Cross Assemblers

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to extend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

Motorola:	6800/6801/6802	now	$32.50
	6805	now	$32.50
	6809	now	$32.50
	68000	now	$50.00
Intel:	8048	now	$32.50
	8051	now	$32.50
	8085	now	$32.50
Zilog:	Z-80	now	$32.50
RCA:	1802/1805	now	$32.50
Rockwell:	65C02	now	$20.00
DEC:	PDP-11/LSI-11	now	$50.00

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

S-C Software Corporation has frequently been commended for outstanding support: competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

S-C Software Corporation (214) 324-2050
P.O. Box 280300, Dallas, Texas, 75228

```
                    1000 *SAVE WETZEL'S PATCHES TO 1.1
                    1010         .OR $F600
                    1020         .TF SCM.PATCH
                    1030 *---------------------------------------------------------
0024-               1040 CH      .EQ $24
0028-               1050 BASL    .EQ $28
0040-               1060 YSAVE   .EQ $40
0200-               1070 WBUF    .EQ $200
C080-               1080 LCPROT  .EQ $C080     LC Protect
C083-               1090 LCWRT   .EQ $C083     LC Write enable
D016-               1100 UCFLAG  .EQ $D016     UC/LC Flag
FF3A-               1110 BELL    .EQ $FF3A     Monitor Bell
                    1120 *---------------------------------------------------------
                    1130 CONTROL.A
F600- C9 81         1140         CMP #$81      Was a CNTL-A entered
F602- D0 12         1150         BNE ERROR     No - then signal error
F604- AD 83 C0      1160         LDA LCWRT     Write enable Language card
F607- AD 83 C0      1170         LDA LCWRT
F60A- AD 16 D0      1180         LDA UCFLAG    Get upper case flag
F60D- 49 FF         1190         EOR #$FF      Reverse it
F60F- 8D 16 D0      1200         STA UCFLAG    Put it back
F612- AD 80 C0      1210         LDA LCPROT    Write protect Language card
F615- 60            1220         RTS
                    1230 ERROR
F616- 20 3A FF      1240         JSR BELL      Ring bell to signal error
F619- 60            1250         RTS           Return
                    1260 *---------------------------------------------------------
                    1270 ESCAPE.C
F61A- E0 00         1280         CPX #0        Start of line?
F61C- D0 1C         1290         BNE .2        No, rtn
F61E- A0 00         1300         LDY #0
F620- B9 3B F6      1310 .1      LDA MSG,Y     Get message
F623- 99 00 02      1320         STA WBUF,Y    Put in buffer
F626- 91 28         1330         STA (BASL),Y  Put on screen (40-column)
F628- C8            1340         INY
F629- C0 07         1350         CPY #7        Finished ?
F62B- D0 F3         1360         BNE .1        Not yet
F62D- 84 40         1370         STY YSAVE
F62F- C8            1380         INY
F630- 84 24         1390         STY CH        Tell assembler
F632- BA            1400         TSX           this was an
F633- A9 CC         1410         LDA #$CC      ESC-L so it will
F635- 9D 03 01      1420         STA $103,X    exec command
F638- A6 40         1430         LDX YSAVE
F63A- 60            1440 .2      RTS
F63B- C3 C1 D4
F63E- C1 CC CF
F641- C7            1450 MSG     .AS -/CATALOG/
```

"One more beep and you're out!"

We have uncovered another neat new Apple accessory: a volume control for the speaker! If other people within earshot of your computer are trying to sleep, or just can't take another five minutes of bells, beeps, and buzzes, the WHISPER VOLUME CONTROL is for you. The Apple version works with II, II Plus, //e, or ///. All you have to do to install it is take the case off your Apple, unplug the speaker wire from the board, plug in the WVC cable, and plug the speaker wire into the other end of the WVC connector. You can also get WVC for the IBM/PC. The retail price is $22.95 for the standard version, or $25.95 with a headphone jack, from Information Dynamics Corp., 1251 Exchange Drive, Richardson, TX 75081. Phone (214)783-8090. Or if you like, buy them from us at $21 and $24, respectively.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).